topology. It queues that task to a second queue Q2 and, if no other tasks are ahead on it, invokes task S1 to effect such processing and updating.

In the meanwhile, SAN service module 38 places on a first queue Q1 further notifications N1', N2', N3', . . . received from the discover engine during processing of a different newly received scan. Upon receiving a scan complete notification for that scan, the service manager creates a task S2 for processing those notifications and updating the manager's SAN topology representation. It queues that task to a second queue Q2 and processes it in order. In the illustrated embodiment, the second queue is a first-in-first-out queue. Thus, task objects S1, S2 are executed in FIFO manner. In alternative embodiments, the second queue may be implemented as a priority queue or otherwise.

Illustrated tasks S1, S2 are represented by respective object-oriented programming (OOP) objects. Each includes method and data members that process the corresponding queued notifications N1, N2, N3, . . . , N1', N2', N3', . . . in FIFO manner. Thus, once an element on the second queue is invoked, the notifications associated therewith on the first queue are processed one at a time by invoking actions, e.g., in the manner discussed above in regard to the policy engine and action automation engine, that, *inter alia*, update the SAN topology maintained by the manager 20 or otherwise accommodate the indicated change.

Though illustrated notifications N1, N2, N3, . . . are processed on a FIFO basis, in alternative embodiments, the notifications of each respective group may be processed based on priority or otherwise with respect to other notifications of the same group. Moreover, though OOP objects

are utilized in the illustrated embodiment, those skilled in the art will appreciate that other constructs may be utilized instead and/or in addition to represent the tasks.

In addition to tasks S1, S2, . . . , that are generated by the service 38 as a result of notifications from the discover engine, further tasks (not shown) may be queued to the task queue Q2 representing operator/administrator requests. These include, for example, requests to change the name of a storage device (e.g., LUN), and so forth. Such tasks are queued in FIFO, priority, or other order, for execution. Unlike the other tasks S1, S2, . . ., the operator/administrator-effected tasks do not involve processing of notifications in the first queue.

This dual approach to handling changes in the SAN, namely, placing asynchronously received scan complete events on a first queue and placing tasks for processing thereof on a second queue allows maintaining a stable representation of various attributes of the SAN, and further ensures that the task notification queues are kept at a reasonable size.

*Conflict Resolution in Event Processing*

Continuing with the above discussion, a task object, e.g., S1, may retrieve further data from the discover engine during processing of its corresponding notifications, N1, N2, N3, . . . . For example, a notification N1 can indicate that a storage device has been added. To update the topology representation maintained by the manager 20, the manager service 38 retrieves the identity of that storage device from the corresponding scan representation maintained by the

discover engine. That information, once obtained, is used by the service 38 to update the topology representation.

In the event the discover engine representation has been modified since the notification N1 was issued, for example, as a result of a later received scan indicating that the newly added storage device was subsequently removed, the manager service 38 detects a logical conflict (e.g., between the event notification N1 indicating that the device has been added and the discover engine database indicating that no such device exists). In such instances, the service 38 employs a conflict resolution mechanism and takes action based on the class of conflict. In the illustrated embodiment, classes of conflicts include modifications of the discover engine representation, e.g., as a result of newly received scans, or corruption of the service manager representation, e.g., as a result of improper action taken on previous events, missed events, database save failures, etc.

Scenarios that indicate corruption and those that indicate a probable change to the underlying representation are identified and documented below. When corruption is absent, no action may be required on the part of the manager service whose goal it is to keep its representation "in sync." However, as a precautionary measure, the manager service can record that an event was received that did not result in an update, and then verify that the expected subsequent event did indeed follow sometime later.

*Handling Events That Appear Inconsistent With Current SAN Manager Services Or Discover Engine Database Contents*